Segunda Avaliação de Linguagens e Compiladores

Pedro d'Aquino – 5434452

Como preparação para a avaliação da disciplina de Linguagens e Compiladores, pede-se para projetar o analisador léxico e sintático de uma linguagem funcional baseada em combinadores chamada "LazyComb".

1. Analisador Léxico

O analisador léxico de um compilador LazyComb é extremamente simples. Não há números, identificadores ou literais; todos os tokens são palavras reservadas da linguagem.

Adotando-se a mesma abordagem utilizada no compilador Cex©, isto é, um autômato finito determinístico para cada classe de tokens, constrói-se um analisador léxico com os seguintes reconhecedores:

- "i"
- "I"
- "s"
- "S"
- "k"
- "K"
- "*"
- "`"
- "("
- ")"
- "0"
- "1"
- Espaço em branco/Comentário

À exceção do último reconhecedor listado, todos retornam um *token* cuja classe é o próprio lexema lido.

A interface do léxico com o resto do compilador dá-se através de três rotinas:

- `HáMaisTokens`: retorna verdadeiro se há mais tokens a serem lidos.
- `OlhaPróximoToken`: retorna o próximo token, sem consumi-lo (*peek*).
- `PegaPróximoToken`: retorna o próximo token e o consome.

A função `OlhaPróximoToken` é utilizada em situações em que é necessário analisar o próximo token para saber qual transição seguir. Na LazyComb, isso só ocorre em uma ocasião (fechamento de parêntesis após uma chamada à submáquina `Expr`).

2. Analisador Sintático

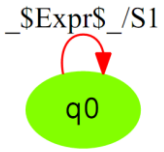
A gramática da LazyComb em notação de Wirth é como segue:

```
Program = { Expr }.
Expr = "i" | "I" | "K" | "k" | "S" | "s" | ( "0" | "1") { "0" | "1" } | "`"
Expr Expr | "*" Expr Expr | "(" { Expr } ")".
```

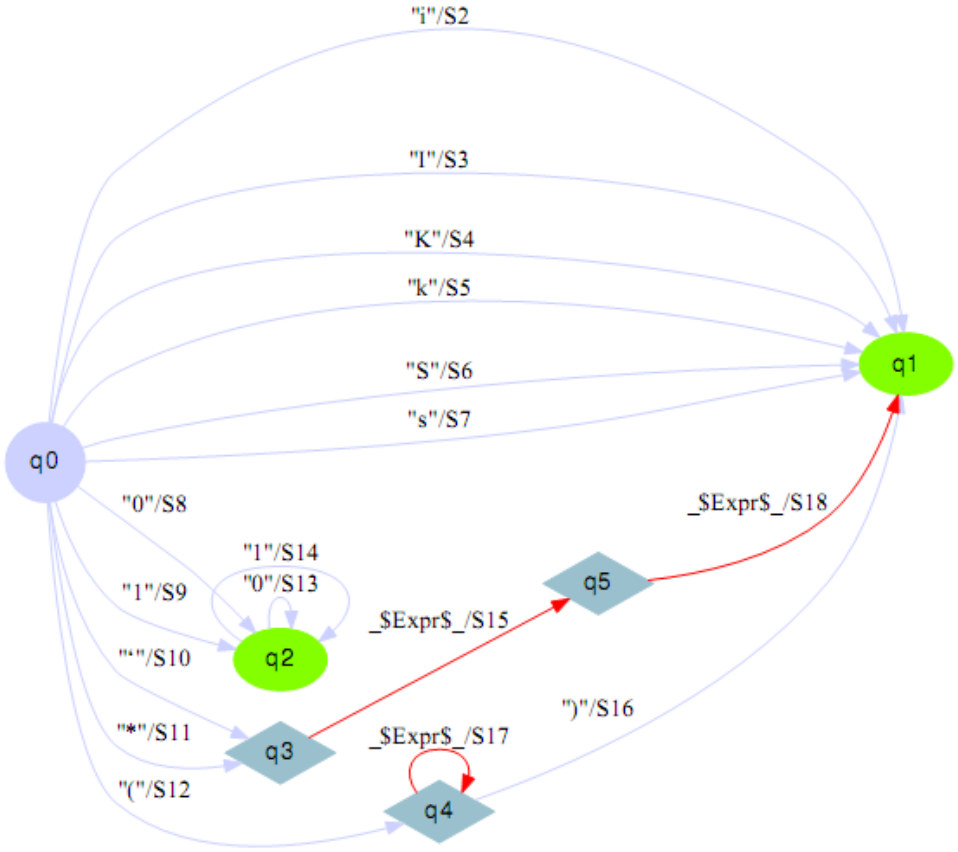
A notação de Wirth é muito mais compacta do que a descrição original em BNF, por dois motivos. Primeira e naturalmente, o poder de expressão de BNF é mais restrito do que notação de Wirth. Além disso, o autor da linguagem quis ressaltar sua relação com quatro outras linguagens funcionais, Iota, Jot, Unlambda e cálculo de combinadores, criando, para isso, produções didáticas. Uma consequência da simplificação realizada é que, ao contrário da gramática original, não há menção aos diferentes significados do terminal “i”. Esse assunto é discutido na seção 2.2.

2.1. Autômatos de Pilha Estruturados

Para reconhecer a linguagem LazyComb, utiliza-se um autômato estruturado de pilha para cada regra na descrição de Wirth apresentada acima. O metacompilador desenvolvido pela Compilex© produz como resultado os seguintes dois autômatos.



Autômato para a regra “Program”



Autômato para a regra “Expr”. O estado inical é “q0”.

Nos diagramas acima, estados coloridos em verde são finais e transições vermelhas são chamadas de submáquina. O nome da submáquina está entre “_$” e “$_”, uma consequência do funcionamento interno do metacompilador. A cada transição está associada uma ação semântica, isto é, uma rotina que será executada durante a

transição. No caso de chamadas de submáquinas, a ação semântica é efetuada após o retorno.

2.2. Dependência de Contexto

LazyComb é *quase* uma gramática livre de contexto. O fato que a torna dependente de contexto é o significado do token "i", que muda dependendo de símbolos anteriores. Especificamente, o comportamento padrão do token "i" é gerar `(lambda (x) x )`. Contudo, quando o compilador recebe um token "*", "i" passa a significar `(lambda (x) (x S K))`. O comportamento original é restaurado quando se encontra "`".

Essa dependência de contexto deve ser tratada em rotinas semânticas nas ações "S10" e "S11" indicadas na figura acima.